Deep Adversarial Bellman Solver for High-Dimensional Heterogeneous Agent Models

Peiyuan Zhang[a]

[a] *School of Economics, Peking University,*

Abstract

Solving high-dimensional Heterogeneous Agent Models (HAMs) is challenged by the curse of dimensionality and limitations of existing methods. We introduce Deep Adversarial Bellman Solver (DABS), a unified deep learning solver directly targeting the Bellman equation using an adversarial framework inspired by Generative Adversarial Networks (GAN). Key innovations ensuring robustness and efficiency include stable multi-shock expectation estimation, constrained data simulation, implicit general equilibrium handling, and experience replay. We validate DABS on the Krusell-Smith model, extensions with continuous/discrete labor supply, and Overlapping Generations (OLG) models, demonstrating its effectiveness, robustness, and broad applicability with the same core algorithm. Our method yields accurate, smooth policy functions, shows improved stability and tail-behavior compared to benchmarks, and provides a powerful, general, and computationally feasible approach for complex HAMs.

Keywords: Heterogeneous Agent Models, Deep Learning, Adversarial Training, Dynamic Programming, Computational Economics, Function Approximation

1. Introduction

In contemporary macroeconomic research, the analytical framework based on heterogeneous agents model (HAM) has gradually established its significance. Agent heterogeneity in income endowments, risk preferences, and life-cycle stages has become a core element in analyzing real-world issues such as wealth distribution, social security policies, and monetary policy transmission mechanisms. The evolution of heterogeneous modeling has been intricately intertwined with the deepening of macroeconomic theory, demonstrating a stepwise expansion of heterogeneity dimensions: Aiyagari (1994) constructed a benchmark analytical framework for heterogeneous savings behavior under incomplete markets by endogenizing individual labor income shocks; Krusell and Smith (1998) further extended the heterogeneity dimensions, achieving the first bidirectional coupling between individual capital accumulation and aggregate productivity shocks, and revealing the asymmetric effects of wealth distribution on shock propagation; the HANK (Kaplan et al., 2018) model characterized micro-level transmission mechanisms of monetary policy shocks by modeling household sector heterogeneity (Kaplan et al., 2018). The deepening of heterogeneous modeling has made explicit the micro-macro interaction mechanisms that traditional representative agent models struggled to capture.

However, solving heterogeneous models faces a fundamental challenge - the curse of dimensionality. When models include multiple heterogeneous state spaces, the dimensionality of the state space grows exponentially with the number of grid points. As shown in Table 1, traditional discretization methods require millions of grid points for 6-dimensional states, with computation time exceeding one week; when the dimensionality increases to 20, the time required would surpass the age of the universe.

Numerous approaches have been proposed to address these challenges: dimensionality reduction techniques such as sparse grids (Smolyak grids) and active subspace methods (Judd et al., 2014); enhancing grid precision through the EGM algorithm (Carroll, 2006); and accelerating solution processes using engineering methods like OpenMP/MPI. However, these methods only alleviate the curse of dimensionality rather than fundamentally resolving it (Maliar and Maliar, 2013; Winberry, 2018).

In recent years, artificial intelligence (AI) technology has provided new insights for overcoming this dilemma. Deep neural networks demonstrate unique advantages through three mechanisms: Implicit dimensionality reduction, Adaptive learning and Nonlinear representation. Notably, Physics-Informed Neural Networks (PINN) have made breakthroughs in solving complex dynamical systems. Raissi et al. (2019) successfully solved highly nonlinear problems like turbulent flow fields by embedding the Navier-Stokes equations into loss functions, verifying the isomorphism between differential equation solutions and deep learning architectures. This technological breakthrough offers significant insights for solving economic models: Bellman equations in macro models essentially form high-dimensional dynamic

State Variables (Dimensions)	Points	Time to Solution
1	10	10 sec
2	100	$\sim$ 1.6 min
3	1,000	$\sim$ 16 mins
4	10,000	$\sim$ 2.7 hours
5	100,000	$\sim$ 1.1 days
6	1,000,000	$\sim$ 1.6 weeks
...	...	...
20	1e20	$\sim$ 3 trillion years

Optimizer		
Dimension reduction	Deal with Points	High-performance computing
Exploit symmetries via the active subspace method	e.g., via (Smolyak/adpative) sparse grids EGM	Reduces time to solution, but not the problem size MPI/OpenMP

Table 1: What is high-dimensional?

programming problems, which share mathematical isomorphism with Policy Evaluation in reinforcement learning - the former requires value functions to satisfy recursive optimality conditions, while the latter approximates optimal policies through temporal difference learning. This isomorphism enables the transfer of deep reinforcement learning techniques such as Generative Adversarial Networks (GAN , Goodfellow et al. (2020)) and Experience Replay to economic model solutions, opening a new path for integrating dynamic programming with deep learning.

Current AI-based discrete-time approaches still have several limitations: Euler equation methods depend on explicit first-order conditions which offer fast convergence but may lack closed-form solutions for problems involving non-convex constraints (such as borrowing limits), discrete choices (like labor participation decisions), or multi-asset portfolios. Bellman equation methods (Maliar et al., 2021) have theoretical universality but face some practical bottlenecks: non-smooth jumps in policy functions, and error amplification in distribution tails. Existing stable solution methods fail to demonstrate the broad applicability, remaining model-specific. This creates a dilemma in discrete-time methods where stable approaches lack model generalization while general methods suffer from convergence instability.

This study achieves systematic breakthroughs based on the framework of Maliar et al. (2021), maintaining one of its core paradigm of solving HAM via Bellman equations. Then innovate the methodological system: First constructs a dual network through parameterized value and policy functions, bypassing Euler equations to directly optimize Bellman residuals; secondly introduces Generative Adversarial Network (GAN) training method, transforming traditional alternating iteration strategies into minmax adversarial optimization mechanisms - the policy network acts as a generator maximizing intertemporal utility to produce candidate decisions, and the value network serves as a discriminator minimizing Bellman residuals to dynamically correct value estimates, forming a self-correcting dynamic equilibrium system; finally addresses policy function instability and non-smoothness issues in original methods using multi-set expectation sampling, prioritized experience replay (Schaul et al., 2015), and simulation-based data generation to stabilize model training.

We propose the Deep Adversarial Bellman Solver (DABS). Compared to original methods, DABS ensures better training stability and generalization. Applying our approach to the traditional Krusell-Smith model resolves tail bias and function smoothness compared to Maliar et al. (2021). Additionally, we validate its generalization by transferring the method to models with labor choices (both continuous and discrete), simple OLG models, and OLG models with housing decisions.

Literature Review. Before applying AI methods to solve HAM, the economics literature traditionally employed three main approaches: one method estimates the evolution of moment conditions for key state variables (Krusell and Smith, 1998; Den Haan, 1997; Fernández-Villaverde et al., 2023); another uses perturbation expansions on aggregate states followed by matrix algebra solutions (Reiter, 2002, 2009, 2010; Winberry, 2018); the third approach applies low-dimensional projections to distributions (Prohl, 2017; Schaab, 2020).

Existing AI-based HAM solutions primarily develop within discrete and continuous time frameworks. Discrete-time models partition decision cycles by time steps, with traditional solutions centered on dynamic programming that iteratively solves optimal decisions period-by-period via value function iteration or policy iteration. Maliar et al. (2021)'s HAM framework solves infinite-horizon problems using simulated data, while Azinovic et al. (2022) develop specialized architectures for Overlapping Generations (OLG) models and Han et al. (2021) solve HAM by period simulation. Continuous-time models directly describe instantaneous state variable changes through differential equations, with initial methods applying deep learning to solve differential equations in continuous-time economic models (Duarte, 2018; Gopalakrishna, 2021; Fernández-Villaverde et al., 2020; Sauzet, 2021). A landmark subsequent paper (Achdou et al., 2022) transformed continuous-time problems into Mean Field Game (MFG) systems (Lasry and Lions, 2007), jointly solving Hamilton-Jacobi-Bellman equations with Kolmogorov forward equations (or their transformed master equations). New literature focuses on solving continuous-time problems within this framework, such as Gu et al. (2023); Payne et al. (2025); Gopalakrishna et al. (2024). Additionally, hybrid approaches exist between discrete and continuous time, primarily using discrete-time approximations to handle forward-backward stochastic differential equations (Han et al., 2018; Huang, 2022). Beyond deep learning, some literature applies reinforcement learning methods to these problems (Hill et al., 2021), while another stream attempts model estimation using deep learning (Kase et al., 2022).

2. Economic Model

We consider a class of dynamic Markov economic models with time-invariant decision functions. An agent (consumer, firm, government, central bank, etc.) solves a canonical intertemporal optimization problem

2.1. Environment

An exogenous shock state $\eta_t \in \mathbb{R}^{n_\eta}$ follows a Markov process with:

$$\eta_{t+1} = M(\eta_t, \varepsilon_t) \tag{1}$$

where ε_t represents i,i,d random disturbances(e.g., technological shocks, labor shocks). This process economically implies that the current macroeconomic state or agent state η_t(e.g., aggregate productivity or personal productivity) and random perturbation ε_t jointly determine the next-period state η_{t+1} , capturing the persistence of shocks.

An endogenous agent state s_t driven by agent decisions x_t (e.g., consumption, investment) and future shocks η_{t+1}:

$$s_{t+1} = S(s_t, \eta_b, x_t, \eta_{t+1}) \tag{2}$$

where x_t depends on personal states s_t, shocks η_t with

$$x_t = X(s_t, \eta_t) \tag{3}$$

An optimization problem can be described as:

$$\mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(s_t, \eta_t, x_t) \right] \tag{4}$$

where$\beta \in (0,1)$is the discount factor reflecting intertemporal preference trade-offs, $u(\cdot)$is the utility function.

2.2. Optimization problem

The agent's decision-making behavior is characterized by the policy function $\psi : R^{N_s} \times R^{N_\eta} \to R^{N_x}$ such that

$$x_t = \psi(s_t, \eta_t) \in X(s_t, \eta_t) \tag{5}$$

Directly finding the optimal function ψ is extremely difficult in high-dimensional problems. To render the optimization problem more tractable, we parameterize the policy function by $\theta_1 \in \Theta_1$, representing the policy function as $\psi(\cdot; \theta_1)$. Here, θ_1 encompasses all the parameters that define the specific form of the policy function. Different values of θ_1 correspond to different decision rules.

After parameterizing the policy function, the original problem of finding the optimal decision rule transforms into an optimization problem of searching for the optimal parameter $\theta_1^* \in \Theta_1$. $\psi(s; \theta_1^*)$, leads to a sequence of decisions $\{x_t^i\}_{t=0}^{\infty}$ starting from any possible initial state (s_0^i, η_s^i), which maximizes the agent's expected lifetime utility:

$$\{x_t^i\}_{t=0}^{\infty} = \arg\max \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t u(s_t, \eta_t, x_t); (s_0^i, \eta_0^i) \right] \tag{6}$$

$$x_t^i = \psi(s_t^i, \eta_t^i; \theta_1^*) \in X(s_t^i, \eta_t^i) \tag{7}$$

Here, $\mathbb{E}$ denotes the expectation taken over all future uncertainties based on the initial information, β is the discount factor reflecting time preference.

Next, we briefly review traditional methods for solving such dynamic programing problems. These methods typically do not directly search for the parameter θ_1 but instead leverage the recursive structure of the problem, particularly the Bellman equation (Value function is parametered by θ_2):

$$V(s, \eta; \theta_2) = \max_{\{x\}} (u(s, \eta, x) + \beta E V(s', \eta'; \theta_2)) \tag{8}$$

where s' is the next-period state obtained according to the state transition function $S(s, \eta, x, \eta')$.

Traditional bellman-based methods such as Value Function Iteration (VFI), iteratively update the value function $V(\cdot)$ and the policy function $\psi(\cdot)$ until convergence which:

Policy Improvement: Given Value function$V(s, \eta; \hat{\theta_2})$, solve

$$x(s, \eta) = \psi(s, \eta; \theta_1) = \arg\max\{u(s, \eta, x) + \beta E[V(s', \eta'; \hat{\theta_2})]\} \tag{9}$$

This step is essentially local optimization - under the existing value function, find the decision rule that maximizes the sum of current and future utilities.

Value Update: Based on latest $\psi(s, \eta; \hat{\theta_1})$, update value function

$$V(s, \eta; \theta_2) = (u(s, \eta, \psi(s, \eta; \hat{\theta_1})) + \beta EV(s', \eta'; \theta_2)) \tag{10}$$

This step substitutes the latest policy function into the Bellman equation and updates the value function by minimizing the Bellman residual.

Convergence Iterate until the difference in value function $\quad V^{(k+1)} - V^{(k)} \quad < \epsilon$. According to the contraction mapping theorem (Stokey and Lucas Jr, 1989), under the condition that the utility function is bounded and the discount factor $\beta < 1$, this iterative process must converge to a unique solution.

2.3. General Equilibrium

Agents make optical decisions based on macroeconomic states (interest rate r and wage w). In a closed economic framework, these aggregate states are not externally imposed but are determined by a large number of heterogeneous agents. To ensure consistency between agent optimal choices and aggregate economic satates, thereby providing a complete description of the entire economic system, we must introduce market clearing conditions and the model's General Equilibrium (GE).

Distribution Dynamics. Let μ_t represent the distribution function of agent states within the population at time t. The evolution of this distribution is controlled by a transition operator Γ:

$$\mu_{t+1} = \Gamma(\mu_t, \psi, \eta_t, \eta_{t+1}) \tag{11}$$

Intuitively, Γ is the set of agent state transition equation $\int S(\cdot)d\mu$.

Aggregate State Update and Market Clearing. The update function of aggregatye state s^{gen}_{t+1}:

$$s^{gen}_{t+1} = \Xi\left(s^{gen}_t, \int s d\mu_t, \int \eta d\mu_t\right) \tag{12}$$

Where Ξ is the market clearance condition, such as the total capital $K_t = \int a_t d\mu_t$, and the interest rate $r_t = \frac{\partial F(\cdot)}{\partial K_t} - \delta$.

Definition of General Equilibrium. A (recursive) General Equilibrium consists of a set of mutually consistent objects: $\{\psi(s,\eta;\theta_1^*), V(s,\eta\theta_2^*), s_{gen}^*\}$, such that:

- Individual Optimization: Given the aggregate states s_{gen}, the agent policy $\psi(s,\eta;\theta_1^*)$ and value function $V(s,\eta;\theta_2^*)$ are the solution to the agent's lifetime utility maximization problem.

- Market Clearing: Aggregate states s_{gen}^* satisify the market clearing conditions Ξ.

- Stationary Distribution: In a steady-state equilibrium, the state distribution μ remains constant under the influence transition operator Γ.

3. Deep Learning Approach

3.1. Adversarial Training: A GAN-Inspired Solution Framework

Generative Adversarial Networks (GAN), introduced by Goodfellow et al. (2020), are a powerful class of deep learning models. Their core idea involves the competition and co-learning of two neural networks: a Generator and a Discriminator, aimed at generating realistic data. The Generator's objective is to learn the distribution of real data and produce "fake" data samples that are indistinguishable from the real ones. The Discriminator's objective is to accurately determine whether an input sample comes from the real dataset or is a "fake" sample generated by the Generator. In this "zero-sum game," the Generator strives to "fool" the Discriminator, while the Discriminator endeavors to "detect" the Generator's fakes. Through this adversarial training process, ideally, the Generator eventually produces highly realistic samples, and the Discriminator can no longer effectively distinguish between real and fake, signifying that the system has reached a Nash equilibrium.

3.1.1. Migrating to Economic Model Solving

Solving dynamic economic models essentially involves finding an optimal policy function ψ and the corresponding value function V that satisfy the Bellman equation. The Bellman equation itself embodies an intrinsic "consistency" or "optimality" check: given a value function V, the optimal policy ψ should maximize the sum of current utility and expected future value; conversely, given a policy ψ, the true value function V should accurately reflect the total expected utility derived from following that policy. This interdependent relationship between "generation" (of policy) and "evaluation" (of value) bears a profound structural similarity (isomorphism) to the adversarial relationship between the Generator and Discriminator in GAN.

We can leverage the adversarial training framework of GANs by transforming the process of finding the optimal policy and value functions into a game between two neural networks.

Policy Network - Playing the Generator Role. The objective of policy network is to generate the "optimal" decision (policy) $\psi(s, \eta; \theta_1)$. "Optimal" here means that, under the evaluation of the current Value Network $V(s, \eta; \hat{\theta_2})$, this policy maximizes the expected lifetime utility. Just as the GAN Generator tries to produce "realistic" samples to fool the Discriminator, the Policy Network attempts to generate decision rules that will receive a "high score" (high expected utility) from the Value Network (the evaluator).

$$\max_{\theta_1}(u(s, \eta, \psi(s, \eta; \theta_1)) + \beta \mathbb{E} V(s', \eta'; \hat{\theta_2})) \tag{13}$$

where s' is the next-period state resulting from the current state s, policy $\psi(s, \eta; \theta_1)$, and next-period shock η'. The expectation $\mathbb{E}$ is taken over the future shock η'. Note that when performing this maximization, the parameters θ_2 of the Value Network V are held fixed.

Value Network - Playing the Discriminator Role. The objective of value network is to accurately estimate the true value of each state (s, η) under a given policy $\psi(s, \eta; \hat{\theta_1})$, which is, to minimize the Bellman residual. The Bellman residual measures the discrepancy between the current value estimate $V(s, \eta; \theta_2)$ and the expected total utility $u+\beta V'$ obtained after following the current policy for one step. Just as the GAN Discriminator tries to accurately distinguish "real" (values satisfying the Bellman equation) from "fake" (values that don't), the Value Network attempts to correctly assess the true long-term value associated with the policy proposed by the Policy Network, rejecting "biased" valuations.

$$\mathcal{L}_{\text{Bellman}} = \Big(V(s, \eta; \theta_2) - \Big(u(s, \eta, \psi(s; \hat{\theta_1})) + \beta \mathbb{E} V(s', \eta; \theta_2)\Big)\Big)^2 \tag{14}$$

When performing this minimization, the parameters θ_1 of the Policy Network ψ are held fixed. The term inside the parentheses, $V - (u + \beta V')$, is precisely the Bellman residual. Minimizing its squared expectation forces the Value Network V to satisfy the Bellman equation.

3.1.2. Overall Objective of Adversarial Training

The entire adversarial training process can be viewed as solving a min-max problem. While the specific formulation might differ slightly from the original GAN, the core adversarial spirit remains. We can interpret it as by:

$$\min_{\theta_2}\max_{\theta_1}\mathcal{L}(\theta_1,\theta_2) = \min_{\theta_2}\left[\underbrace{V(s,\eta;\theta_2) - \underbrace{\max_{\theta_1}(u(s,\eta,\psi(s,\eta;\theta_1)) + \beta E V(s',\eta';\theta_2))}_{\text{Maximize utility}}}_{\text{Minimize Bellman residual}}\right]^2 \tag{15}$$

Here, the objective function $\mathcal{L}$ aims to find (ψ, V) pairs that satisfy the Bellman equation. One interpretation is that $\max_{\theta_1}$ drives the policy network to find high-return policies, while $\min_{\theta_2}$, by minimizing the Bellman error, penalizes value estimates inconsistent with the dynamics, thereby indirectly constraining the "feasibility" or "truthfulness" of the final policy.

3.2. Neural Networks as Universal Approximators

In the previous subsection, we introduced the GAN-inspired adversarial training framework, which solves the Bellman equation through the competition between a Policy Network (Generator) and a Value Network (Discriminator). To implement this framework, we require powerful function approximation tools capable of representing the high-dimensional, non-linear policy function ψ and value function V.

We choose to approximate the policy ψ and the value function V using Deep Neural Networks (DNNs) instead of conventional methods like polynomial functions or other interpolation techniques. Neural networks offer significant advantages when dealing with high-dimensional economic models, particularly those involving complex interactions and constraints, making them a preferable choice:

- Linear Scalability: Traditional grid-based methods or polynomial interpolation suffer from the "curse of dimensionality," where computational and memory requirements grow exponentially with the state space dimension. In contrast, the number of parameters in a neural network typically grows only linearly with the dimensionality.

- Robustness and Feature Learning: Neural networks exhibit good robustness to multicollinearity among input features. More importantly, through their hierarchical structure (hidden layers), they can automatically learn underlying structures and correlations within the data, performing effective feature extraction or implicit dimensionality reduction, capturing complex relationships between state variables (see section 3.3.3).

- Flexibility for Fitting Non-linear Environments: Economic models often feature complex non-linearities, such as kinks arising from constraints (e.g., borrowing limits, non-negativity), discontinuities or discrete choices resulting from institutions or decisions (e.g., labor force participation), and potential regime switching. Neural networks, especially those employing non-linear activation functions (like ReLU), are exceptionally well-suited for fitting such highly non-linear functions, enabling them to capture these intricate economic behaviors.

The basic structure of a neural network consists of interconnected cells. Each cell receives inputs from other cells or environment, processes them (typically via a weighted matrix $\theta_i^{(1,2)}$ followed by an activation function $a_{1,2}$), and transmits the processed signal to other connected cells . A typical multi-layer network (see Figure 1) includes an input layer, one or more hidden layers, and an output layer. The hidden layers extract more abstract and useful representations of the information by applying a series of non-linear transformations to the input signals, endowing neural networks with greater expressive power and flexibility compared to methods like polynomials that establish a direct input-output relationship.

Leveraging the powerful approximation capabilities of neural networks, we replace the key functions in our theoretical model—the policy function ψ and the value function V with specific neural network architectures:

Policy Network $\psi(s, \eta; \theta_1)$. This network takes the state variables (s, η) as input and outputs the decision variable x. Its parameters are determined by

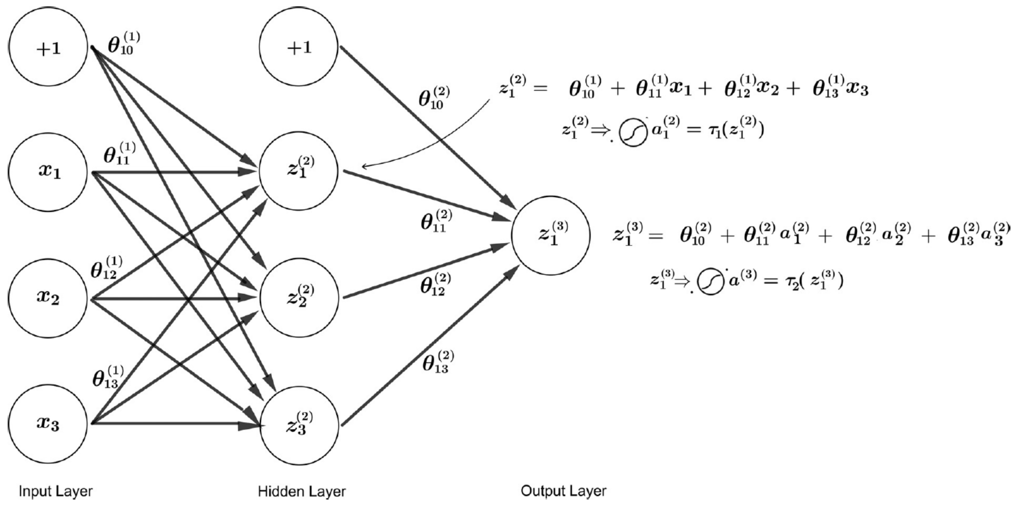

+1
x_1
x_2
x_3
$\theta_{10}^{(1)}$
$\theta_{11}^{(1)}$
$\theta_{12}^{(1)}$
$\theta_{13}^{(1)}$
+1
$z_1^{(2)}$
$z_2^{(2)}$
$z_3^{(2)}$
$\theta_{10}^{(2)}$
$\theta_{11}^{(2)}$
$\theta_{12}^{(2)}$
$\theta_{13}^{(2)}$
$z_1^{(3)}$
$z_1^{(2)} = \theta_{10}^{(1)} + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3$
$z_1^{(2)} \Rightarrow a_1^{(2)} = \tau_1(z_1^{(2)})$
$z_1^{(3)} = \theta_{10}^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}$
$z_1^{(3)} \Rightarrow a^{(3)} = \tau_2(z_1^{(3)})$
Input Layer
Hidden Layer
Output Layer

Fig. 1: A neural network with one hidden layer.

θ_1 (containing weights W_i^η and biases b_i^η in i th layer). For example, if the policy is a consumption rate with $\frac{\text{consumption}}{\text{wealth}} \in [0,1]$, a Sigmoid activation function can be used in the output layer to ensure the output satisfies the constraint:

$$\psi(s,\eta;\theta_1) = \text{Sigmod}\left(W_2^\eta \cdot \text{Tanh}(W_1^\eta[s,\eta] + b_1^\eta) + b_2^\eta\right) \tag{16}$$

Where NN has only one hidden layers and typically use activation functions like Tanh to introduce non-linearity.

Value Network $V(s,\eta;\theta_2)$. This network also takes the state variables (s,η) as input and outputs an estimate of the value for that state. Its parameters are determined by θ_2. Since value functions are often unbounded, a linear activation function is commonly used in the output layer:

$$V(s,\eta;\theta_2) = W_2^V \cdot \text{ReLU}(W_1^V[s,\eta] + b_1^V) + b_2^V \tag{17}$$

By selecting appropriate network architectures and activation functions, NNs can be designed to automatically satisfy economic constraints. For instance, in a model with non-negativity wealth constraint $w \geq 0$ and consumption constraint $c \in [0,w]$, parameterizing the decision as the consumption ratio $\frac{c}{w}$ and using a Sigmoid output automatically ensures $\frac{c}{w} \in [0,1]$, thereby satisfying $0 \leq c \leq w$ without needing to handle these boundary conditions manually during optimization.

3.3. Training Optimization and Stability Techniques

3.3.1. All-in-One (AIO) Expectation and Paired Shock Sampling

When handling the expectation term $E[V(s', \eta')]$ in the Bellman equation for dynamic models, exact computation is typically infeasible, necessitating Monte Carlo methods. The All-in-One (AIO) concept introduced by Maliar et al. (2021) approximates this expectation by drawing a single pair of independent shocks, balancing computational efficiency and estimation accuracy. We extend this method by adopting a Multi-Pair Shock Sampling strategy to further enhance the stability and accuracy of the estimation. Specifically, instead of being limited to a single shock pair, we independently draw M pairs of future shock. Using multiple shock pairs provides a more robust expectation estimate and reduces the variance of the Monte Carlo estimation, thereby greatly improving the stability of the training process. Thus we have:

$$\max_{\theta_1} \left(u(s, \eta, \psi(s, \eta|\theta_1)) + \beta \frac{1}{M} \sum_{m}^{M} (V_{\epsilon^{\eta}_{1,m}}(s', \eta'|\theta_2) + V_{\epsilon^{\eta}_{2,m}}(s', \eta'|\theta_2)) \right) \tag{18}$$

$$\mathcal{L}_{\text{Bellman}} = \frac{1}{M} \sum_{m}^{M} [\left(V(s, \eta; \theta_2) - \left(u(s, \eta, \psi(s, \eta; \hat{\theta}_1)) + \beta V_{\epsilon_{1,m}}(s', \eta'; \theta_2) \right) \right) \tag{19}$$

$$\times \left(V(s, \eta; \theta_2) - \left(u(s, \eta, \psi(s, \eta; \hat{\theta}_1)) + \beta V_{\epsilon_{2,m}}(s', \eta'; \theta_2) \right) \right)] \tag{20}$$

Although using M pairs requires more computation compared to just one pair, which might have been challenging due to computational limitations when the original method, we overcome this bottleneck by leveraging the powerful parallel processing capabilities of modern GPUs.

3.3.2. Constrained Parallel Simulation for Data Generation

To supply high-quality data that aligns with the model's intrinsic logic for neural network training, we employ a simulation-based data generation method. The core of this approach involves using the currently trained policy network ψ to drive the dynamic evolution of the economy, thereby generating (state, policy, next state) samples. Instead of simulating a single representative economy, we simulate k independent virtual economies in parallel, each

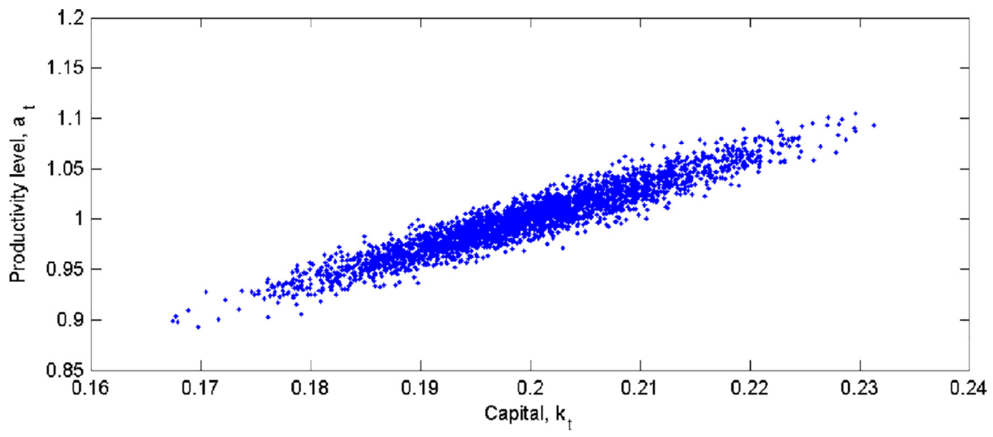


Fig. 2: Ergodic set of a neoclassical growth model.

populated with ℓ heterogeneous agents. At each time step, agents make decisions according to the rules dictated by the policy network, and their states transition based on relevant economic laws or model equations, such as the budget constraint.

A crucial implementation detail is the strict enforcement of these economic constraints during the simulation process, ensuring all generated sample points reside within the economically feasible domain. This avoids meaningless or impossible states that could arise from random sampling and guarantees data validity. The primary advantage of this simulation-based method, compared to random sampling detached from model dynamics, is that the generated data naturally reflects the model's dynamic properties and endogenous interactions, concentrating on the regions of the state space that agents actually experience (i.e., the model's ergodic set, see Figure 2).

3.3.3. Implicit Market Clearing via Neural Networks

For models requiring a general equilibrium solution, we employ the strategy of implicit market clearing encoded within the neural network to avoid the costly nested loops typical of traditional methods. By directly including key aggregate information (such as aggregate capital K, distribution moments $\int a d\mu$, aggregate shocks z) alongside agent states as inputs to the policy and value networks, and optimizing within the end-to-end adversarial framework, the neural network automatically learns the dependence of agent decisions on aggregate variables and how the macro state influences the agent's environment (e.g., interest rate). This design implicitly embeds market clearing conditions into the learning process, allowing for the simultaneous coordination of micro-level optimization and macro-level consistency within a single network pass and update, thereby substantially enhancing solution efficiency.

3.3.4. Experience Replay

To overcome issues related to temporal correlations in sequentially simulated data, improve sample efficiency, and enhance training stability, we incorporate an Experience Replay mechanism. State transition data generated during simulations are continuously stored in a dynamic memory pool. During training, mini-batches are randomly sampled from this pool for gradient updates. This random resampling breaks the temporal dependencies of the data, leading to more stable training and reducing short-term policy oscillations. Furthermore, by mixing data stored from different historical periods and economic states, experience replay enhances the model's generalization capabilities, enabling it to learn from a more diverse set of experiences. This mechanism also effectively decouples the data generation and model training steps, facilitating parallel data processing and model updates on GPUs, thus accelerating the overall training pipeline.

4. Application

4.1. Krusell-Smith Mode

4.1.1. Model Setup

The model features an economy composed of ℓ heterogeneous agents who share identical fundamental preferences but differ in their productivity levels and asset holdings.

Each agent i chooses consumption c_t^i and beginning-of-next-period assets (determined by savings $k_t^i = w_t^i - c_t^i$) to maximize their expected lifetime CRRA utility:

$$\max_{\{c_t^i, k_{t+1}^i\}_{t=0}^{\infty}} E_0 \left[\sum_{t=0}^{\infty} \beta^t \frac{(c_t^i)^{1-\gamma} - 1}{1-\gamma} \right]$$

$$s.t.\ w_{t+1}^i = R_{t+1} k_{t+1}^i + W_{t+1} e^{y_{t+1}^i}$$

$$c_t^i \leq w_t^i$$

Here, w_t^i is agent i's cash-on-hand/wealth at the beginning of period t, y_t^i is their log labor productivity, R_t is the gross return on assets $(1+r_t)$, and W_t is the wage rate. Agent provides one unit of labor and theit log productivity y_t^i follows an exogenous AR(1) process:

$$y_{t+1}^i = \rho_y y_t^i + \sigma_y \epsilon_{t+1}^i, \quad \text{with } \epsilon_{t+1}^i \sim N(0,1) \tag{21}$$

A representative firm uses aggregate capital K_t and aggregate effective labor L_t for production via a Cobb-Douglas function:

$$F(K_t, L_t) = z_t K_t^\alpha L_t^{1-\alpha} \tag{22}$$

Aggregate productivity z_t follows an exogenous (apparently linear) AR(1) process:

$$z_{t+1} = \rho_z z_t + \sigma_z \epsilon_{t+1}, \quad \text{with } \epsilon_{t+1} \sim N(0,1) \tag{23}$$

Aggregate capital K_t is the sum of all agent end-of-period savings (or beginning-of-period assets) $K_t = \sum_i^\ell k_t^i$, and aggregate effective labor L_t is the sum of individual effective labor units $L_t = \sum_i^e ll \exp(y_t^i)$. Factor prices are determined by marginal products (δ is the depreciation rate):

$$R_t = 1 - \delta + z_t \alpha K_t^{\alpha-1} L_t^{1-\alpha} \tag{24}$$

$$W_t = z_t (1-\alpha) K_t^\alpha L_t^{-\alpha} \tag{25}$$

4.1.2. State Space and Neural Network Input:

Theoretically, an agent's decision depends on their own state (w_t^i, y_t^i) and the aggregate state of the economy, which is jointly determined by the distribution of all agent states $D_t = \{(w_t^i, y_t^i)\}_i^\ell$ and the aggregate productivity shock z_t. To utilize neural networks for the solution, we feed this high-dimensional state information as input. Specifically, for agent i, the input vector h_t^i to the neural network comprises:

- Current state information of all agents: $D_t = \{(w_t^i, y_t^i)\}_i^\ell$.
- The current aggregate shock: z_t
- Agent i's own specific state: (w_t^i, y_t^i).

Thus, for each agent i, the input layer dimension is $2\ell + 1 + 2 = 2\ell + 3$.

4.1.3. Neural Network Approximations

We use neural networks to approximate the key functions:

- Policy Function (Consumption Ratio): $\psi(s_t^i;\theta_1)$ outputs the ratio of consumption to cash-on-hand $\frac{c_t^i}{w_t^i}$, using a Sigmoid function to ensure it lies within $[0,1]$.

$$\frac{c_t^i}{w_t^i} = \sigma(\zeta_0^\psi + \iota^\psi(y_t^i, w_t^i, D_t, z_t; \upsilon^\psi)) = \psi(s_t^i;\theta_1) \tag{26}$$

- Value Function: $V(s_t^i;\theta_2)$ directly outputs the estimated value of state s_t^i.

$$V_t^i = \zeta_0^V + \iota^V(y_t^i, w_t^i, D_t, z_t; \upsilon^V) = V(s_t^i;\theta_2) \tag{27}$$

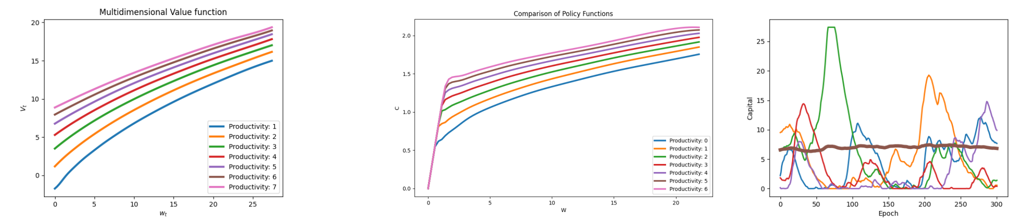
Here, $\iota()$ represents the neural network (with parameters υ), ζ_0 is a learnable adjustment coefficient, and $\theta = \{\zeta_0, \upsilon\}$ denotes all parameters for that network. These networks share the same input structure h_t^i.

4.1.4. Pesudo Code

See Algorithm 1

4.1.5. Results

We applied our proposed deep learning solver to the Krusell-Smith model and compared the results with benchmark methods from the literature (Maliar et al., 2021), while also analyzing the training dynamics and robustness of our approach.



(a) Value function (b) Policy function (c) Simulation

Fig. 3: Baseline

Algorithm 1 Bellman-based training - Pretrain Phase

```
 1: Neural Network Settings
 2: $\Phi^P(S;\Theta^P)$: Policy network
 3: $\Phi^V(S;\Theta^V)$: Value network
 4: $S^i = (\{w^j, y^j\}_{j\in L}, z, w^i, y^i)$        ▷ Agent state representation
 5: Initialization
 6: Initialize $\Theta_0^P, \Theta_0^V$ with normal initialization
 7: Initialize memory pool $MP \leftarrow \emptyset$
 8: Draw initial economy states $Data_0^{sim} \in \mathbb{R}^{Batch_{sim}\times L\times 2}$
 9: for $k = 1$ to $Epochs_{pretrain}$ do
10:     Pretrain Value Network
11:     for $iter = 1$ to $train\_value\_times\_each\_step$ do
12:         Sample batch $\{w^i, y^i\}_{i=1}^L \sim MP$
13:         Compute target $\hat{v}^i = u(c^i) + \beta E_M[\Phi^V(S^{i'};\Theta^V_{fixed})]$
14:         Update $\Theta^V$: $\min \frac{1}{L}\sum_{i=1}^L \frac{1}{M}\sum_{m=1}^M[(\Phi^V(S^i;\Theta^V) - \hat{v}^i_m)]$
15:     end for
16: end for
17: for $k = 1$ to $Epochs_{total}$ do
18:     for $phase = 1$ to $alternating\_steps$ do
19:         Policy Network Update
20:         for $iter = 1$ to $train\_policy\_times\_each\_step$ do
21:             Sample batch $\{w^i, y^i\}_{i=1}^L \sim MP$
22:             Compute $c^i = \Phi^P(S^i;\Theta^P)$
23:             Compute gradient: $\nabla_{\Theta^P}\mathbb{E}_M[u(c^i) + \beta\Phi^V(S(c^i);\Theta^V_{fixed})]$
24:             Update $\Theta^P$ using Adam
25:         end for
26:         Value Network Update
27:         for $iter = 1$ to $train\_value\_times\_each\_step$ do
28:             Sample new batch $\{w^i, y^i\}_{i=1}^L \sim MP$
29:             Compute $\hat{v}^i = u(\Phi^P(S^i;\Theta^P_{fixed})) + \beta E_M[\Phi^V(S^{i'};\Theta^V)]$
30:             Update $\Theta^V$: $\min \frac{1}{L}\sum_{i=1}^L \frac{1}{M}\sum_{m=1}^M[(\Phi^V(S^i;\Theta^V) - \hat{v}^i_m)]$
31:         end for
32:     end for
33:     Memory Pool Update
34:     for $sim = 1$ to $simulate\_times$ do
35:         Maintain queue: $MP \leftarrow Data_{new} \in f(Data_{current}, \Phi^P)$
36:     end for
37: end for
```

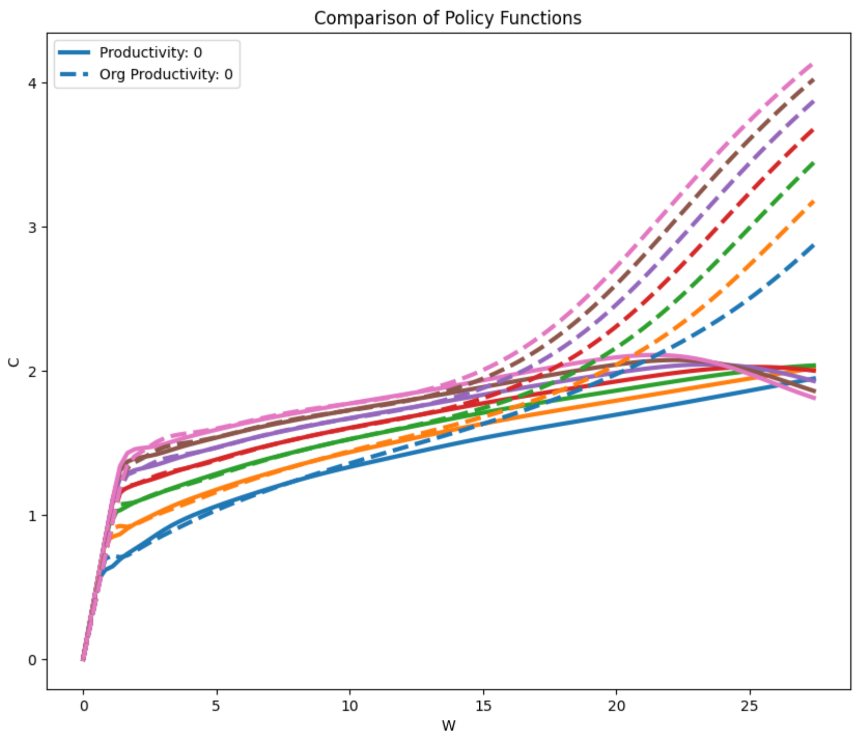


Fig. 4: Compared policy functions

Overall Solution Quality and Steady-State Simulation. From a macroscopic perspective, our method successfully solves the Krusell-Smith model. The obtained policy function ψ (as shown in Figure 3b) exhibits economically plausible behavior (e.g., consumption rate decreases with assets, increases with productivity). More importantly, when this policy function is used for long-run simulations, the economy's key aggregate variables (such as aggregate capital) display stable dynamics, eventually fluctuating around a steady state (as depicted in Figure 3c). This is qualitatively consistent with theoretical expectations and existing literature, validating the fundamental effectiveness of our solver.

Improvements in Policy Function Accuracy and Smoothness. A detailed comparison with the results of Maliar et al. (2021) (shown in Figure 4) reveals

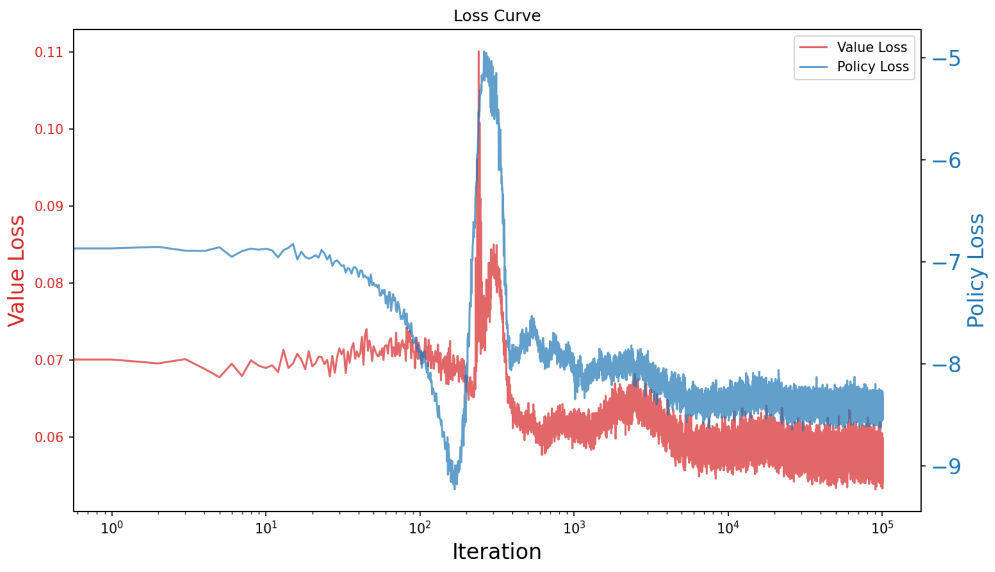
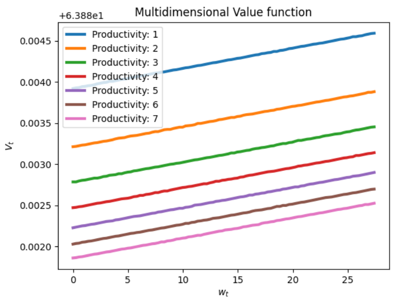


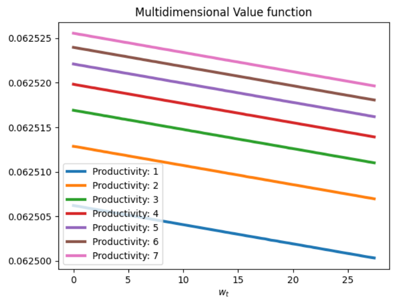
Fig. 5: Compared policy functions

advantages of our policy function in several key aspects. Firstly, our method significantly improves the accuracy and stability of the policy function in the tail regions of the state , avoiding potentially large errors or implausible behavior in these areas. Secondly, our policy function exhibits greater overall smoothness, particularly in the lower asset range (front end), effectively mitigating the non-smooth kinks or jumps that can arise in traditional or some other deep learning-based methods.

Training Losses and Adversarial Equilibrium. Given that our method is based on GAN-inspired adversarial training, the dynamics of the loss functions during training (illustrated in Figure 5) differ from those in standard supervised learning. The objective for the policy and value networks is not necessarily to drive the loss to zero, but rather to reach an Adversarial Equilibrium through mutual competition. As the figure shows (note that the x-axis often uses a logarithmic scale, log10(training iterations), to display long-term trends), the loss functions of both networks might exhibit significant fluctuations in the early stages of training, reflecting the ongoing ”game” and mutual adaptation between them. As training progresses, the losses gradually stabilize. This stabilization precisely indicates that the policy and value networks have reached a relatively stable equilibrium state, signifying that a solution approximately satisfying the Bellman optimality conditions has been found.



(a) Random initial 1



(b) Random initial 2

Fig. 6: Different intialization

Method Robustness Check. To examine the stability and sensitivity of our solver to initial conditions, we conducted multiple training runs, each time using a different random seed to initialize the neural network parameters. The results demonstrate (as shown in Figure 6) that despite starting from different initial parameters, all training runs consistently converged to essentially the same policy function and economic steady state. This indicates that our method possesses good robustness and its solution is insensitive to the random initialization of the neural networks, bolstering our confidence in the reliability of the obtained results.

4.2. Model Extension: Endogenous Labor Supply Choice

To further test the generalizability of our proposed adversarial training framework, we apply it to extensions of the standard Krusell-Smith model that incorporate endogenous labor supply decisions. The significance of testing generalizability lies in the fact that an ideal solver should adapt to different model structures without requiring substantial modifications to its core training algorithm or loss functions. Notably, while Maliar and Maliar (2022) also studied models with discrete labor choices, they employed different training strategies and loss function definitions compare with Maliar et al. (2021). In contrast, we utilize the exact same adversarial training framework and optimization objectives as used for the baseline KS model to directly solve extended models featuring both continuous and discrete labor choices, thereby validating the broad applicability of our method.

4.2.1. Model Setup

We modify the baseline Krusell-Smith model to allow agents to choose their labor supply l_t.

Continuous Labor Supply. Agents can now continuously choose their labor supply time l_t within the interval $[0, 1]$. The utility function is accordingly modified to include the disutility of labor:

$$u(c,l) = \frac{(c^{\kappa}(1-l)^{1-\kappa})^{1-\gamma} - 1}{1-\gamma} \tag{28}$$

where $\kappa > 0$ control the ratio between leisure $(1 - l)$ and consumption c. The labor income term in the budget constraint now becomes w_t^i, η_t^i, l_t^i:

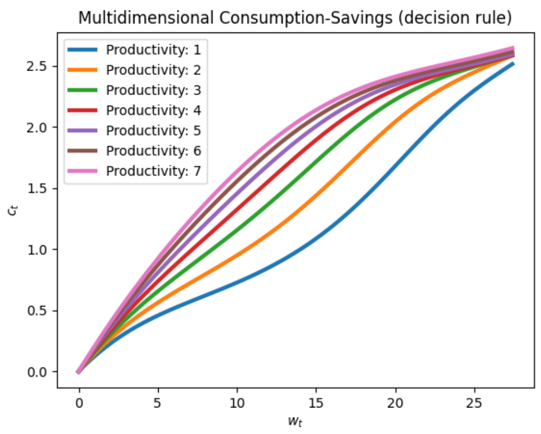
$$w_{t+1}^i = R_{t+1}(w_t^i - c_t^i) + W_{t+1} l_t^i e^{y_{t+1}^i} \tag{29}$$

For the neural network implementation, the output dimension of the policy network increases, needing to output both the consumption decision and the labor supply decision l_t. Since l_t is constrained to [0, 1], a Sigmoid activation function can be used for the output corresponding to l_t.

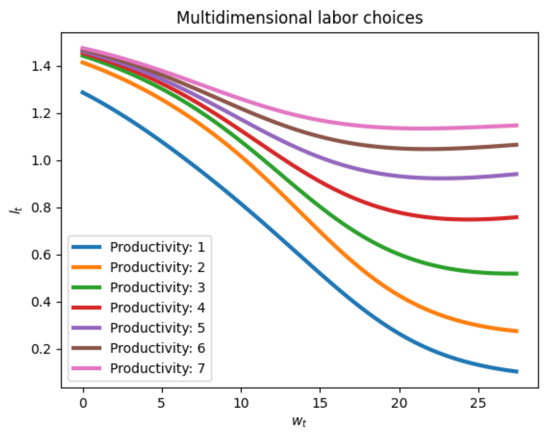
Discrete Labor Supply. An alternative scenario, involves individuals making labor participation decisions from a finite set of options, for example, $l_t \in \{0, 1\} or \{0, 0.5, 1\}$ (representing unemployed, part-time, and full-time). We employ the Gumbel-Softmax trick at the end of the policy network. In this case, the policy network outputs the probabilities of choosing each discrete labor option, rather than the level itself.

4.2.2. results

We successfully applied the unmodified adversarial training framework to both of the aforementioned labor supply extension models. The results demonstrate (see Figures 7 (continuous), Figure 8 (binary discrete), Figure 9 (ternary discrete) for details) that the method effectively solves these more complex models. Whether dealing with continuous labor supply or discrete scenarios with two or three choices, the training process remained stable, and the resulting policy functions (for consumption, savings, and labor supply) exhibited economically plausible properties. For instance, the intertemporal substitution effect in labor supply was clearly captured in the continuous model. This strongly supports the generality of our proposed unified framework, showcasing its ability to handle different types of decision variables and model structures without requiring specific algorithmic adjustments.

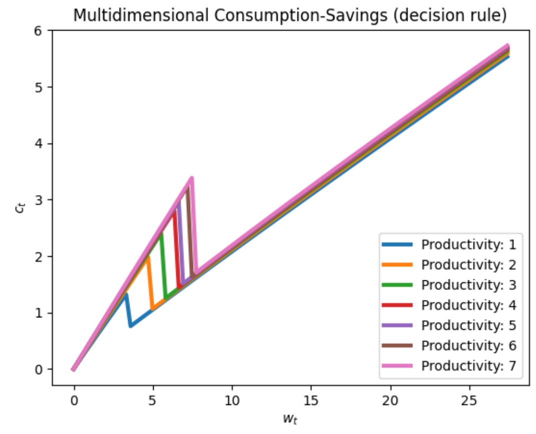


(a) Consumption choice

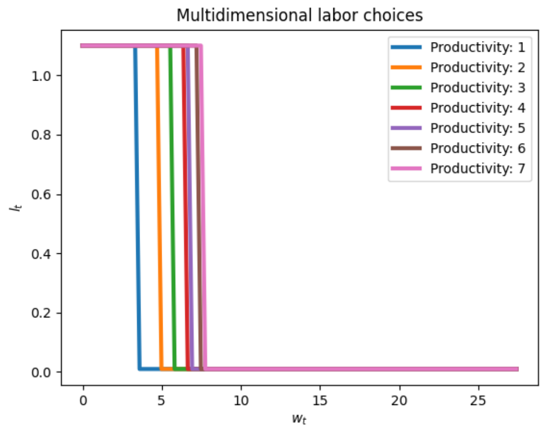


(b) Labor choice

Fig. 7: Continous Labor choice model

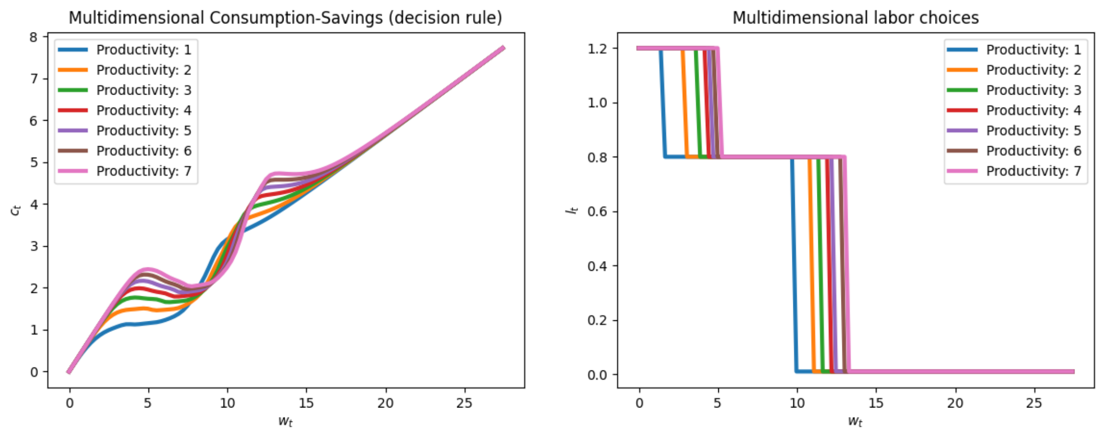


(a) Consumption choice



(b) Labor choice

Fig. 8: 2 Labor choices model



(a) Consumption choice (b) Labor choice

Fig. 9: 3 Labor choices model

4.3. OLG Models

To further showcase the adaptability of our framework, we apply it to Overlapping Generations (OLG) models. Unlike infinite horizon models, agents in OLG models have finite lifespans, passing away after reaching a maximum age, while new agents are continuously born. This core feature necessitates certain adjustments to our original framework, particularly in the definition of the value function, the neural network design, and the methods for generating and utilizing training data.

4.3.1. Methodological Adjustments

Age Dependency and Value Function Structure Adjustment. The most critical adjustment is incorporating age j (from 1 to the maximum age J) as part of the state variables. Consequently, both the value function and policy function explicitly depend on age: $V(s, \eta, j; \theta_2)$ and $\psi(s, \eta, j; \theta_1)$. The neural network's input layer is expanded accordingly to include age j (e.g., via one-hot encoding) as an additional input. More importantly, the finite lifespan introduces a definitive Terminal Condition: agents in their final period of life J have no future, and their value is solely determined by their current period:

$$V(s, \eta, J; \theta_2) = u(c_J) \tag{30}$$

This terminal condition requires special handling during neural network training, typically meaning that gradients are not computed for the value function output of the final age group.

Target Value Calculation via Backward Iteraton. Since the solution to OLG models inherently follows a Backward Iteraton structure, we must adhere to this logic when training the value network V to compute its target value V_{target}. Specifically, given the current policy network $\psi(;\theta_1)$, to calculate the target value for an agent of age j in state (s, η), we need to simulate forward this agent's path from age $j+1$ to the end of life J and compute the discounted sum of all future period utilities:

$$V_{\text{target}}(s, \eta, j) = \sum_{t=j+1}^{J} \beta^{t-j} u(s_t, \eta_t, t, \psi(\cdot|\hat{\theta}_1)) \tag{31}$$

The value network's loss function is then defined based on this backward-iteration-calculated target value:

$$\mathcal{L}_{Bellman}(s, \eta, j) = (V(s, \eta, j|\theta_2) - V_{\text{target}}(s, \eta, j))^2 \tag{32}$$

This ensures the value network learns the true lifecycle value profile consistent with theory.

Data Generation Adjustment. The data generation process needs to reflect the demographic structure of the OLG model. We simulate N economies in parallel, each containing all J Age Cohorts at any point in time, with each cohort comprising ℓ agents of the same age. The simulation dimension thus increases, requiring tracking of $N \times \ell \times J$ agents. Furthermore, the simulation must include a birth and death mechanism: at the end of each period, the current oldest cohort (age J) "dies" and is removed from the simulation, all other agents age by one ($j \to j+1$), and ℓ new agents of age 1 are "born" (typically with specific initial states, like zero assets). This simulation approach (illustrated in Figure 10) ensures that the data in the experience replay pool covers all stages of the lifecycle.

4.3.2. A Simple OLG Model Example

To demonstrate the adapted method, we consider a simple OLG model in Azinovic et al. (2022).

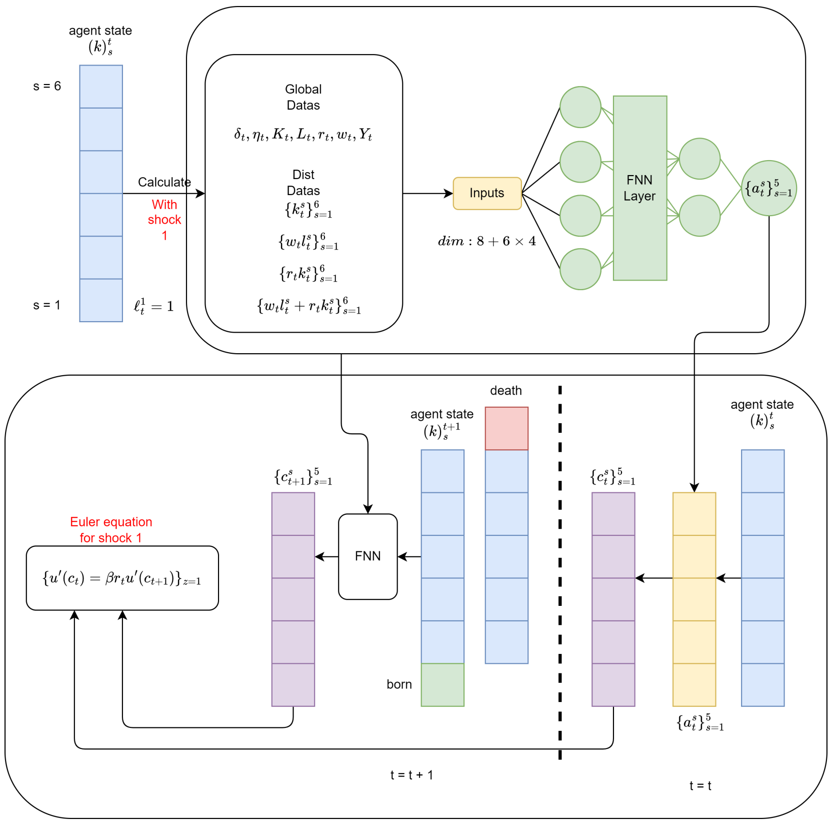


Fig. 10: Simulating with OLG

Household. Household live for $A = 6$ periods, period t, household age s. Household supply labor $l_t^0 = 1$ only in new born with:

$$\begin{aligned}
\max_{\{c_{t+i}^i, a_{t+i}^i\}_{i=0}^{A-1}} E_t \sum_{i=0}^{A-1} \log(c_{t+i}^i) \\
c_t^h + a_t^s = r_t k_t^s + w_t l_t^h \\
k_{t+1}^{s+1} = a_t^s \\
a_t^{A-1} \geq 0
\end{aligned}$$

Firm. With C-D form product function

$$\begin{aligned}
f(K_t, L_t, z_t) &= \eta_t K_t^\alpha L_t^{1-\alpha} + K_t(1-\delta_t) \\
K_t &= \sum_{s=0}^{A-1} k_t^s \\
L_t &= \sum_{s=0}^{A-1} l_t^s = 1 \\
r_t &= \alpha \eta_t K_t^{\alpha-1} L_t^{1-\alpha} + (1-\delta_t) \\
w_t &= (1-\alpha)\eta_t K_t^\alpha L_t^{-\alpha}
\end{aligned}$$

with and shock with $\{1, 2, 3, 4\}$i.i.d, accordingly $\delta_t \in \{0.5, 0.5, 0.9, 0.9\}$, $\eta_t \in \{0.95, 1.05, 0.95, 1.05\}$.

4.3.3. Results

We applied the adapted adversarial training framework to this simple OLG model. Figure 11 shows a comparison between the policy function derived by our method (e.g., savings rate by age, represented by dots) and a benchmark solution (e.g., obtained via traditional numerical methods or an analytical solution under specific conditions, represented by circles). As observed, the solution simulated by the neural network closely approximates the benchmark solution, indicating that our framework adjustments were successful in accurately capturing the lifecycle dynamics inherent in the OLG model.

5. Conclusion

The accurate analysis of high-dimensional Heterogeneous Agent Models (HAMs) is crucial for modern macroeconomics, yet poses significant computational challenges due to the curse of dimensionality. Traditional numerical

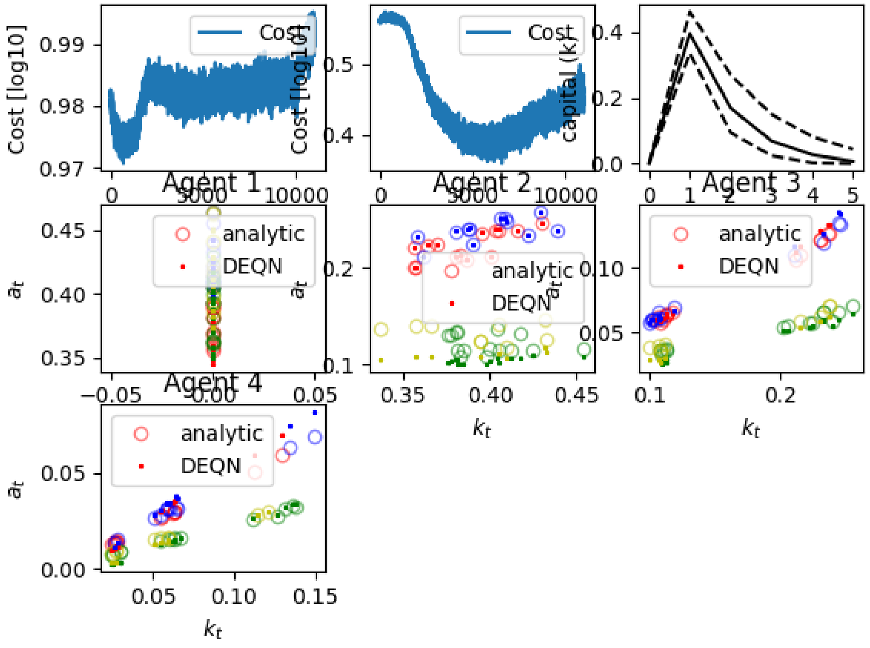


Fig. 11: Basic OLG

methods often struggle with high dimensions, while existing Artificial Intelligence (AI)-driven approaches, despite their potential, are frequently limited by reliance on specific equation forms (e.g., Euler equations), training instability, or a lack of generalizability across diverse model structures.

Addressing these limitations, this paper proposed and implemented a unified, Bellman equation-based deep learning solver (Adversarial Dynamic Programming Deep Solver, DABS). Our core contribution lies in constructing an adversarial training framework inspired by Generative Adversarial Networks (GANs). Within this framework, a parameterized policy network (acting as the generator) and a value network (acting as the discriminator) compete to jointly find the equilibrium solution satisfying the Bellman optimality conditions. This approach directly targets the Bellman equation, circumventing the need for models to possess explicit first-order conditions (Euler equations).

To ensure the framework's effectiveness, stability, and efficiency, we integrated several key technical innovations: multi-pair shock sampling accelerated by GPUs optimizes the stability and efficiency of expectation calculations; an economically constrained parallel simulation data generation mechanism ensures the quality and relevance of training data; implicit encoding of market clearing conditions within the neural network enables the method to efficiently handle models with general equilibrium features without extra outer loops; and the introduction of an experience replay mechanism breaks temporal correlations in data, enhancing training stability and sample efficiency.

Through validation across a range of standard and extended models, we demonstrated the effectiveness and broad applicability of our framework. On the benchmark Krusell-Smith model, our method not only accurately solved the model but also outperformed certain benchmark methods in terms of policy function smoothness and behavior in tail regions. More importantly, we showcased the framework's generalizability: using the identical core adversarial training algorithm, we successfully solved extended models featuring both continuous and discrete endogenous labor supply choices, without needing to modify the training strategy or loss function for different models, unlike approaches in some related literature. Furthermore, with adaptive adjustments to the value function definition and training process, the framework effectively solves Overlapping Generations (OLG) models characterized by finite lifecycles. Results from multiple runs with different initializations also confirmed the robustness of our method.

In summary, the proposed deep learning-based adversarial Bellman solver provides a powerful, general-purpose, and computationally feasible tool for tackling complex, high-dimensional heterogeneous agent models. By directly and stably solving the Bellman equation, it lowers the barrier to analyzing economic models involving non-smooth constraints, discrete choices, or intricate feedback loops, offering a promising pathway for advancing the frontiers of quantitative macroeconomic research. Future research directions could include further enhancing computational efficiency, exploring more sophisticated network architectures, and applying the framework to cutting-edge models like HANK with richer heterogeneity dimensions (e.g., multi-asset portfolio choice).

References

Achdou, Y., Han, J., Lasry, J.M., Lions, P.L., Moll, B., 2022. Income and wealth distribution in macroeconomics: A continuous-time approach. The review of economic studies 89, 45–86.

Aiyagari, S.R., 1994. Uninsured idiosyncratic risk and aggregate saving. The Quarterly Journal of Economics 109, 659–684.

Azinovic, M., Gaegauf, L., Scheidegger, S., 2022. Deep equilibrium nets. International Economic Review 63, 1471–1525.

Carroll, C.D., 2006. The method of endogenous gridpoints for solving dynamic stochastic optimization problems. Economics letters 91, 312–320.

Den Haan, W.J., 1997. Solving dynamic models with aggregate shocks and heterogeneous agents. Macroeconomic dynamics 1, 355–386.

Duarte, V.F., 2018. Machine learning for continuous-time economics .

Fernández-Villaverde, J., Hurtado, S., Nuno, G., 2023. Financial frictions and the wealth distribution. Econometrica 91, 869–901.

Fernández-Villaverde, J., Nuno, G., Sorg-Langhans, G., Vogler, M., 2020. Solving high-dimensional dynamic programming problems using deep learning. Unpublished working paper .

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2020. Generative adversarial networks. Communications of the ACM 63, 139–144.

Gopalakrishna, G., 2021. Aliens and continuous time economies. Swiss Finance Institute Research Paper .

Gopalakrishna, G., Payne, J., Gu, Z., 2024. Asset pricing, participation constraints, and inequality. Participation Constraints, and Inequality (November 8, 2024) .

Gu, Z., Laurière, M., Merkel, S., Payne, J., 2023. Deep learning solutions to master equations for continuous time heterogeneous agent macroeconomic models. Technical Report.

Han, J., Jentzen, A., E, W., 2018. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences 115, 8505–8510.

Han, J., Yang, Y., et al., 2021. Deepham: A global solution method for heterogeneous agent models with aggregate shocks. arXiv preprint arXiv:2112.14377 .

Hill, E., Bardoscia, M., Turrell, A., 2021. Solving heterogeneous general equilibrium economic models with deep reinforcement learning. arXiv preprint arXiv:2103.16977 .

Huang, J., 2022. A probabilistic solution to high-dimensional continuous-time macro and finance models. Available at SSRN 4122454 .

Judd, K.L., Maliar, L., Maliar, S., Valero, R., 2014. Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain. Journal of Economic Dynamics and Control 44, 92–123.

Kaplan, G., Moll, B., Violante, G.L., 2018. Monetary policy according to hank. American Economic Review 108, 697–743.

Kase, H., Melosi, L., Rottner, M., 2022. Estimating nonlinear heterogeneous agents models with neural networks. Centre for Economic Policy Research.

Krusell, P., Smith, Jr, A.A., 1998. Income and wealth heterogeneity in the macroeconomy. Journal of political Economy 106, 867–896.

Lasry, J.M., Lions, P.L., 2007. Mean field games. Japanese journal of mathematics 2, 229–260.

Maliar, L., Maliar, S., 2013. Envelope condition method versus endogenous grid method for solving dynamic programming problems. Economics Letters 120, 262–266.

Maliar, L., Maliar, S., 2022. Deep learning classification: Modeling discrete labor choice. Journal of Economic Dynamics and Control 135, 104295.

Maliar, L., Maliar, S., Winant, P., 2021. Deep learning for solving dynamic economic models. Journal of Monetary Economics 122, 76–101.

Payne, J., Rebei, A., Yang, Y., 2025. Deep learning for search and matching models. Available at SSRN 5123878 .

Prohl, E., 2017. Discetizing the infinite-dimensional space of distributions to approximate markov equilibria with ex-post heterogeneity and aggregate risk .

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics 378, 686–707.

Reiter, M., 2002. Recursive computation of heterogeneous agent models. manuscript, Universitat Pompeu Fabra , 28–35.

Reiter, M., 2009. Solving heterogeneous-agent models by projection and perturbation. Journal of Economic Dynamics and Control 33, 649–665.

Reiter, M., 2010. Approximate and almost-exact aggregation in dynamic stochastic heterogeneous-agent models .

Sauzet, M., 2021. Projection methods via neural networks for continuous-time models. Available at SSRN 3981838 .

Schaab, A., 2020. Micro and macro uncertainty. Available at SSRN 4099000.

Schaul, T., Quan, J., Antonoglou, I., Silver, D., 2015. Prioritized experience replay. arXiv preprint arXiv:1511.05952 .

Stokey, N.L., Lucas Jr, R.E., 1989. Recursive methods in economic dynamics. Harvard University Press.

Winberry, T., 2018. A method for solving and estimating heterogeneous agent macro models. Quantitative Economics 9, 1123–1151.